A Hybrid Column Generation approach for an Industrial Waste Collection Routing Problem

Hauge, Kristian ; Larsen, Jesper; Lusby, Richard Martin ; Krapper, Emil

Published in:
Computers & Industrial Engineering

Link to article, DOI:
10.1016/j.cie.2014.02.005

Publication date:
2014

Document Version
Early version, also known as pre-print

A Hybrid Column Generation approach for an Industrial Waste Collection Routing Problem

Kristian Hauge[a], Jesper Larsen[b,*], Richard Martin Lusby[b], Emil Krapper[a]

[a] *Transvision, Hejrevej 34 D, 3., Copenhagen, Denmark*
[b] *Department of Management Engineering, Technical University of Denmark, Produktionstorvet, building 426, 2800 Kgs. Lyngby, Denmark*

Abstract

This paper presents a practical roll-on/roll-off routing (ROROR) problem arising in the collection of industrial waste. Skip containers, which are used for the waste collection, need to be distributed between, and collected from, a set of customers. Full containers must be driven to dump sites, while empty containers must be returned to the depot to await further assignments. Unlike, the traditional ROROR problem, where vehicles may transport one skip container at a time regardless of whether it is full or not, we consider cases in which a vehicle can transport up to eight containers, at most two of which can be full. We propose a Generalized Set Partitioning formulation of the problem and describe a hybrid column generation procedure to solve it. A fast Tabu Search heuristic is used to generate new columns. The proposed methodology is tested on nine data sets, four of which are actual, real-world problem instances. Results indicate that the hybrid column generation outperforms a purely heuristic approach in terms of both running time and solution quality. High quality solutions to problems containing up to 100 orders can be solved in approximately 15 minutes.

Keywords: Routing, Rollon-Rolloff, Waste collection, Column Generation, Metaheuristic

1. Introduction

This paper focuses on a routing problem that arises in connection with the disposal of bulky waste using large containers. In a complex world where environmentally friendly solutions and recycling are on the top of the agenda, waste management systems become even more complicated. This has forced municipalities to prioritize and implement cost-effective solutions to deal with all kinds of waste. Here, we consider waste that comes from industry and which must be transported to dump sites using containers.

This particular problem belongs to the roll-on/roll-off routing (ROROR) class of problems that already exist in the literature. The ROROR problem is a variant of the very general framework of the Rich Vehicle Routing Problem (RVRP) – see eg. **? ?**]. As we will see, the ROROR problem can be specialized further depending on the different characteristics and constraints of the problem.

*Corresponding author
Email addresses: kha@transvision.dk (Kristian Hauge), jesla@dtu.dk (Jesper Larsen), rmlu@dtu.dk (Richard Martin Lusby), ekr@transvision.dk (Emil Krapper)

We will in this paper look at a specific variant of the RORORproblem in order to demonstrate how optimization-based methods and metaheuristics together can result in efficient problem solving and a flexible framework. Pratical RORORproblems are today solved using very simple heuristics approaches because the constraints and characteristics make an exact approach challenging to implement and the solution time potentially intractable. In addition, all RORORproblems are subtly different and therefore a framework that can be modified is necessary, and this flexibility is difficult to get with an exact approach. Companies that build software for the waste management industry therefore rely on a range of flexible heuristic framework. Finally, we also see this paper as a vehicle to push the use of optimization-based methods and metaheuristics to a wider range of vehicle routing problems than just the RORORproblem. Many of the problems within the general definition of Rich Vehicle Routing problems would be interesting to study using framework developed in this paper.

Simple extensions include a maximum number of trucks, many types of goods (different types of containers), capacity constraints and multiple depots. The more complex features of the problem include the introduction of disposal facilities as well as four different order types, each requiring several visits at depots, customers and/or dump sites.

?] gives an introduction to waste collection as a vehicle routing problem component of the overall waste management process. A classification of RORORproblems into residential, commercial and industrial is also provided. Whereas residential problems are mainly viewed as arc routing problems, the RORORproblem is in general seen as a vehicle routing problem where nodes are used to represent depots, dump sites and customers. The RORORproblem is often characterized by a number of different *trip types*, which together comprise a complete tour for a vehicle. As an example, the following sequence of trip types would describe a complete tour for a truck starting and ending at the same depot: the truck leaves depot, it drives empty to a customer, here it picks up a container, the truck then empties the container at a dump site and transports the empty container back to customer before returning to its depot.

To the best of the authors' knowledge, the first paper defining the RORORproblem is **?**]. Here, a problem set up with a single depot and a single dump site is presented, and four different heuristics are devised based on seeing the problem as a combined vehicle routing and bin packing problem.

The RORORproblem we consider closely resembles the problem studied by **?**]; the order types are practically the same, there are different sizes and shapes of containers, and the available dump sites and depots vary depending on which order is considered. However, one major difference exists. Namely, the capacity. In the problem presented by **?**], each vehicle can transport at most one container. It is therefore not possible to mix the visits related to different orders. As soon as a vehicle has picked up a container for one order, it cannot attend another order before it has delivered the container in question, at which point the first order is completed. The problem presented by **?**] considers using vehicles with a capacity of two containers and is concerned with the collection of containers that are being scraped. The authors describe an enumeration approach for generating a large set of routes.

?] refer to the problem as the *skip collection* problem and study several interesting features such as different waste types, multiple dump sites, priorities and time windows. Again the solution approach is based on a heuristic algorithm. In addition, **?**] consider a skip collection problem with industrial as well as domestic customers and four different trip types. As in most papers within the area, the capacity of a

vehicle is assume to be one container.

More recent papers within the area are **? ?**]. The authors propose metaheuristic solution approaches and add more realistic constraints like time windows, changing service types and heterogeneous vehicle fleet to the RORORproblem. The problem instances contain between 50 and 200 orders for their instances.

Our main contribution is to describe and implement a solution approach for a variant of the RORORwhere multiple containers can be stacked on top of each other, and, in addition, we focus on an approach that combines the exact approach of column generation with advanced metaheuristics. This gives a solution approach that exploits the benefits of state-of-the-art exact approaches for RVRP's with the flexibility of metaheuristics.

The rest of the article is structured as follows. In Section 2, we define the problem considered in more detail and include a review of existing literature on related problems. Section 3 presents the model we propose and discusses the devised solution approach. The algorithm is tested extensively in Section 4, where comparisons are made between the developed algorithm and the purely heuristic approach currently being used by our industrial partner. Finally, conclusions and directions for future work are summarized in Section 5. The main contributions of this paper is twofold; first we describe a solution approach to a real-life routing problem and secondly we describe a hybrid approach between exact methods and heuristic approaches.

2. Problem description

As mentioned previously, the problem under consideration deals with transportation of bulky waste containers. A problem instance is defined by a set of *orders*, a set of *locations*, and a set of *trucks* that can be used for handling the orders. An order consists of *picking up* and/or *delivering* and/or *emptying* a container at a specific location, which can be one of the three following types. A *customer* location refers to the order of a certain customer, whose location is the geographical location of the customer. A *dump site* refers to the place where containers must be taken for emptying. Full containers must be taken to a site before they can be taken anywhere else. Note that it is not permitted to leave an empty container at a dump site. Finally, a *depot* denotes the location where empty containers are stored and collected from.

There are four types of orders in the problem, all of which involve visiting some or all of the location types defined above. The first type is termed the *pickup* order. This entails picking up a full container of waste from one of the customers and transporting it to a dump site. The empty container is then returned to a depot.

A *delivery* order is defined similarly. It simply entails delivering an empty container to a particular customer.

A combination of a pickup order and a delivery order is termed a *swap*. Here, an empty container is picked up at a depot and transported to the customer. At the customer the empty container is put down to replace a full container. The full container is picked up during the same visit and taken to a dump site to be emptied. It is then returned to a depot.

A so-called *change* order resembles the pickup order; however, instead of taking the empty container back to a depot, it must be returned to the customer from whom it was picked up earlier. Figure 1 gives an example of this. Such an order is used if there is not enough free space at the customer for performing a swap, or if the company does not have ownership over the container.

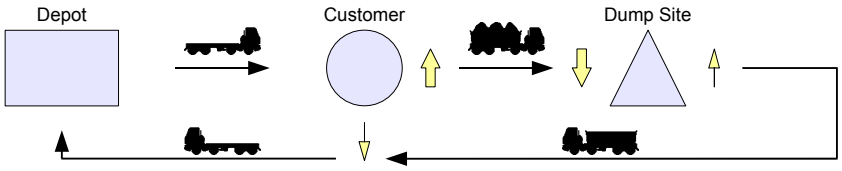

Figure 1: A change order.

Despite the fact that bulky waste containers are large, some trucks can carry more than one at a time. Orders that involve visiting the same locations can be handled simultaneously to save time and money. An example of this is illustrated in Figure 2, where two customers are located close to each other, but far from the dump site. Here it is advantageous to visit both customers prior to visiting the dump site.

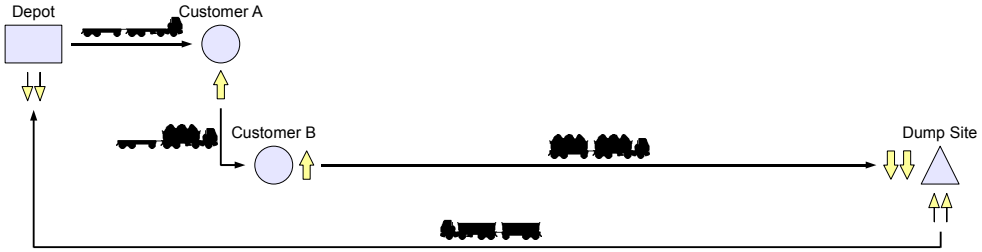

Figure 2: Two closely located pickup orders.

We now briefly describe several features, and extensions, that concern dump sites, depots, the capacity of the trucks, and the order structure. In many vehicle routing problems (VRPs), each order corresponds to a single visit, namely a visit to the customer who placed the order. In this problem, any order consists of several *sequenced* visits. For example, the swap order consists of picking up an empty container at a depot (visit 1), delivering the empty container and picking up a full container at the customer (visit 2), emptying the full container at a dump site (visit 3), and returning the now empty container to a depot (visit 4). As is shown in Figure 2, visits belonging to the same customer do not necessarily have to be scheduled immediately after each other; the vehicle is allowed to visit customer B between two visits that are related to order A (the visit at customer A and the visit at the dump site). Furthermore, the visit at the dump site in Figure 2 is actually the dump site visit for both orders A and B since the containers of both these orders are emptied during the visit. Even though visits of the same order are not required to be performed immediately after each other, they must be performed by the same vehicle and in the correct order. This means that if the first visit of a swap order is assigned to some vehicle, then that vehicle will also have to perform visits 2, 3 and 4 of that order later on its route. For customer visits, we assume that we can pick up (and/or deliver) the container at any time during the working hours of a typical day. That is, customers do not have individual time windows which must be respected.

Containers come in different types. When a customer places an order, the order concerns a specific type of container. Furthermore, if visits are planned in a clever way, the vehicles might not have to return all the emptied containers to the depots. When a container has been emptied as part of either a pickup order or a swap order

(but not a change order), it can be delivered to another customer who has placed either a delivery order or a swap order, instead of being returned to a depot. This way, the last two visits of one swap order (or of a pickup order) could be merged with the first two visits of another swap order (or of a delivery order), but *only* if the container types of the two orders are the same, see Figure 3. This maneuver will be referred to as *annihilation*.

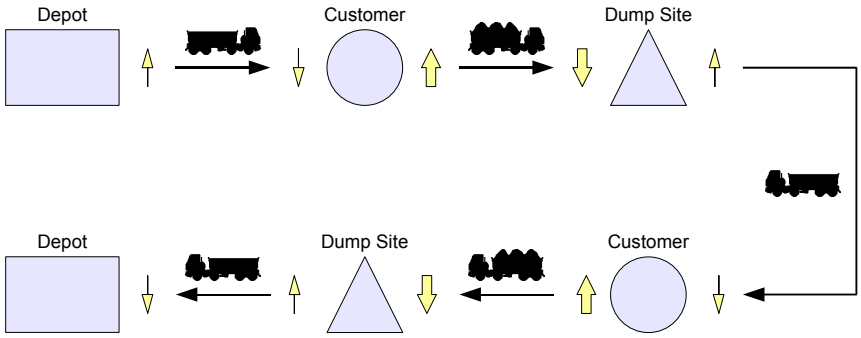

Figure 3: The annihilation principle. Two consecutive swap orders connected by annihilation. The empty container from the first customer is delivered to the second customer

There are many depots and many dump sites. For simplicity, we assume that any dump site or depot can handle any container type and, furthermore, that there are no limitations on opening times. As is the case with customers, we instead observe a working day, where any depot/dump site can be visited at any time within the specified working hours. After a container has been picked up from a customer it must be returned to a depot (or another customer) after being emptied.

Additionally, one must respect an upper limit on how many containers a vehicle can transport at any given time. It's capacity can be increased if it can tow a trailer. In this work we assume a homogeneous vehicle fleet, where each vehicle also has a trailer. Each vehicle can hence carry up to two stacks of containers at a time; one on the vehicle itself, and an additional one on the trailer. A stack may consist of up to four containers of which only one may be full (the one on the top of the stack). Therefore, in total each vehicle can carry up to eight containers at a time, and no more than two of these containers may be full. From a stacking perspective, we also assume that any stacking combination is possible; in reality this is unlikely to be the case due to the different shapes and sizes of the containers.

We solve the problem for one day at a time. For such a day, routes are created for all the vehicles such that all orders for that day are covered by these routes. All routes start and end at a specific depot, which is called the *main depot*. A main depot is specified for each problem instance. The different visits of an order may not be divided between different routes. Each route consists of a number of visits at different locations. For each visit it is specified what should be done during that visit in terms of pickups and deliveries, exactly what types of full and empty containers should be picked up and put down during the visit, and to which orders the containers relate. The goal is to design the routes such that the total cost of the selected routes is as low as possible. The cost of a route for this problem is defined as the total time it will take the vehicle to cover the route and complete all the tasks assigned to that route. Here driving time between two points is calculated assuming a constant vehicle speed of $16\frac{m}{s}$ using the straight line distance between the respective locations. This will also

be referred to as the *value* of the route. The term *route length* will be used to denote the number of orders on a route. The total distance that is traveled on a route will simply be referred to as the *total distance*.

Average *handling times* are used for handling the containers at the customers. Each type of container has a pickup time and a delivery time (in terms of duration) associated with it, both for full and empty containers. These four handling times can be considered as average handling times for that container type. When estimating the time spent delivering and picking up containers at some location, only the container types are relevant and not their positions in the stacks. This means that putting down the container on top of a stack will take the same amount of time as putting down the one in the bottom of the stack, assuming that they are both empty. Full containers are always on top.

We consider the problem as described above. It includes a number of interesting features. For example, orders consisting of multiple sequenced visits that must be performed by the same vehicle. In addition, we also consider the annihilation principle and two interdependent capacities.

Naturally, there are several interesting extensions to the problem. We include a brief description of these here. Firstly, one may look at including a more sophisticated measure of driving time. One could also look at not only introducing depot/dump site specific opening hours, but also include restrictions on which dump sites can process which container types and which container types can be stored at which depots. Similarly, time windows stating when it is possible to visit a customer could be included. Finally, it would be interesting to consider rules regarding possible stacking configurations of the containers on vehicles. For example, a large container cannot be placed on top of a small one but a small container can be placed on top of (inside) a large one. It is not only the size of a container that matters, but also the shape. If the vehicle is carrying a full container, it must be on the top of a stack. By omitting these issues we have a more general yet still realistic problem.

We conclude this section with an example. Figure 4 shows two solutions to a problem that contains one of each type of order and three different types of containers. The purpose of this figure is also to illustrate how two seemingly good solutions can be significantly different in terms of their costs. The arrows that represent pickups and deliveries now have different colors to represent different types of containers. Trucks are replaced by containers in this figure so that stacks of containers can be illustrated easily. The total time spent handling containers is the same for the two solutions, but driving distance is not. Using Euclidean distances between the centers of the locations, the total driving distance of Solution 1 is 11% longer than that of Solution 2.

3. Solution approach

RORORO problems have typically been solved using heuristic approaches. The main reason is the large and complex set of constraints and characteristics. This makes an exact approach very difficult and, even if possible, potentially intractable in time given that the allowed planning horizon is often minutes for real-life RORORO problems. In addition, due to the real-life nature of the problem, the flexibility in a heuristic framework rather than a less-flexible (exact) mathematical model is often appreciated by software companies as well as the end-users of the planning tools.

The solution approach is as follows: A hybrid Column Generation (CG) heuristic solves the problem by combining the functionalities of two different optimization

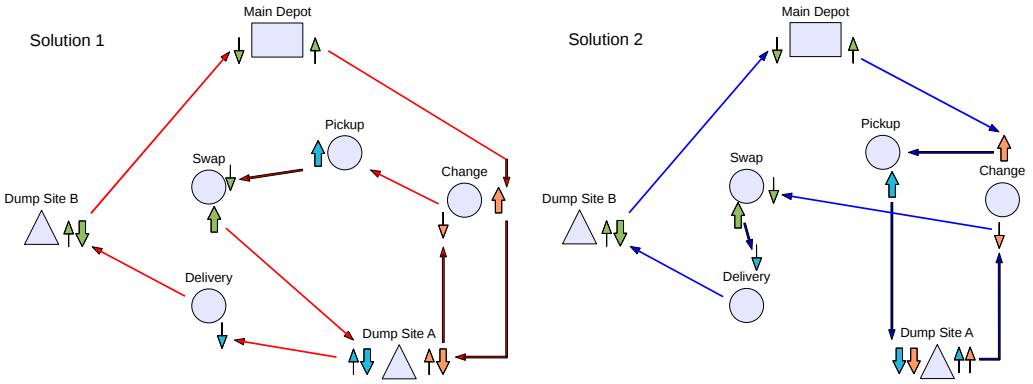

Figure 4: Example solutions

methods, namely CG and Tabu Search (TS). Both these optimization methods could in theory be used alone for solving the problem but are here combined to exploit the best of both parts. CG has shown excellent results for routing problems and with a heuristic pricing problem, we can in an easy and efficient way maintain a high level of flexibility for the problem. The popular metaheuristic TS is chosen because it has produced good results for VRPs in many cases; for example, **?**] and **?**]. Since the RORORproblem can be formulated in a way that is suited for a CG scheme. Solving the subproblem to optimality would produce the column with the most negative reduced cost in each iteration. Optimality comes with a price as this often leads to very long running times.

The RORORproblem can be described as the following integer programming problem. Let O be the set of orders for an instance and let R be the set of all feasible routes. In addition a_{or} is equal to 1 if route r contains order o and is 0 otherwise, and c_r is the cost of using r in the solution. Finally, N is the number of trucks in the instance. This gives the following generalized set partitioning model:

$$\min \sum_{r \in R} c_r x_r \tag{1}$$

$$\text{s.t.} \sum_{r \in R} a_{or} x_r = 1 \qquad \forall o \in O \tag{2}$$

$$\sum_{r \in R} x_r \leq N \tag{3}$$

$$x_r \in \{0;1\} \qquad \forall r \in R \tag{4}$$

The cost of a route is defined by measuring the time it takes to complete the route. First define the sets C as the set of all container types. Each change order has its own container type. Furthermore let O be the set of all orders and V the set of all customers to reflect that we can have multiple orders at a customer, and finally let S be the set of sites. Time during the route is used on driving from i to j denoted t_{ij} and handling time of the containers. Handling time of containers is based on container type, if it is empty or full and whether we are picking it up or delivering it. Handling times for a container of type c are denoted: picking up full ($h_c^{p,f}$), picking up empty ($h_c^{p,e}$), delivering full ($h_c^{d,f}$) and delivering empty ($h_c^{d,e}$). In order to describe the objective function we have the following binary variables: Let x_{ij} equal 1 if the

route goes directly from i to j. Furthermore, let p^f_{ic} and p^e_{ic} denote whether we pick up a full respectively empty container of type c at location i or not, and d^e_{ic} if we deliver an empty container of type c at location i. Finally d^f_{sc} specifies whether we deliver a full container of type c at dump site s or not. Now we can write up the function defining the cost of a route (column):

$$c_r = \sum_{c \in C} \left(h^{f,p}_c \sum_{i \in V} p^f_{ic} + h^{f,d}_c \sum_{i \in V, s \in S} d^f_{sc} + h^{e,p}_c \sum_{i \in V} p^e_{ic} + h^{e,d}_c \sum_{i \in V} d^e_{ic} \right) + \sum_{i,j \in V^2} (t_{i,j} x_{i,j}) \tag{5}$$

Next (2) secures that each order is carried out while (3) ensure that we do not use more than N trucks. Clearly the model requires a total enumeration of all feasible routes for the problem in order to be able to solve it to optimality. An alternative to generating all routes a priori is to use (dynamic) column generation by first solving the LP relaxation of a restricted problem. The restricted master problem (RMP) becomes:

$$\min \sum_{r \in R'} c_r x_r \tag{6}$$

$$\text{s.t.} \sum_{r \in R'} a_{or} x_r = 1 \qquad \forall o \in O \tag{7}$$

$$\sum_{r \in R'} x_r \leq N \tag{8}$$

$$x_r \in [0;1] \qquad \forall r \in R' \tag{9}$$

where R' is only a subset of the feasible routes. We can then solve this relaxation to establish values of the dual variables of the constraints (7) and (8) for the pricing problem. Then by solving the pricing problem we find out if there are any columns (routes) with negative reduced cost then they are added to the RMP. We then resolve the RMP and can make another iteration, if no routes with negative reduced cost was found we have found the LP optimum. Implementing this in a branch-and-bound framework results in a branch-and-price algorithm.

The idea to solve the pricing problem using heuristics is not new. Since the pricing problem is often computationally expensive to solve, heuristics have been suggested and used earlier in the literature as eg. **?** **?**], but usually the exact pricing problem has always been solved to prove optimality (see eg. **?**]). In this paper we only solve the pricing problem using our TS.

3.1. Solving the pricing problem Using Tabu Search

Every time the pricing problem is to be solved, the TS algorithm is initialized and the dual vector from the restricted master problem is given as input. The objective is to find a new route with a reduced cost as low as possible. Based on the definition of c_r in (5) we can define the objective function of the pricing problem. Let y_o be equal to 1 if order o is part of the route and let γ be the dual variable corresponding to (8) and let α_o be the dual variables corresponding to (7). The objective function of the pricing problem can be described as:

$$\min c_r - \gamma - \sum_{o \in O} y_o \alpha_o \tag{10}$$

Each route will consist of a sequence of visits at different locations. In the RORORproblem when visiting a customer, it is well known what should happen during that visit, depending on the order type. This is not the case for visits at depots and dump sites where a variable number of containers can be picked up, delivered, or emptied, depending on the current load of the truck. In order to be able to modify solutions at this level, the TS heuristic will treat a solution as a sequence of visits. The algorithm will need to be able to modify all these specifications in order to be able to design the best routes.

Defining the Tabu Search

The objective function of the pricing problem will be almost the same as presented earlier for the c_r coefficients in the master problem. This objective function is an expression of the reduced cost of the route, which is exactly the measure of interest. The *neighborhood function* defines how the algorithm searches for new solutions. Since a solution to this pricing problem is a route where up to 20 customers (an approximate upper bound derived from the real-life instances that have been made available to us) are serviced, the neighborhood of a solution is defined as the set of all routes that can be obtained by adding or removing one order to or from the current route.

More Neighbors

In most cases, adding or removing an order gives rise to a number of decisions to be made. Using a pickup order as an example, there are three visits to be carried out. Firstly, it must be decided when to visit the customer and pick up the full container. Secondly, it must be decided when to empty that container, and there may be more than one dump site to choose from. Lastly, the empty container should be returned to one of many depots or delivered to some other customer who has placed either a delivery order or a swap order for a container of that exact type. The latter option entails a new annihilation. Clearly, adding an order to an existing route could result in many different neighbors. Up to four visits are to be included on the route resulting in a lot of different combinations.

However, many of these combinations will not be feasible after all. The precedence constraints governing the ordering of these visits cannot be violated; the container must be picked up before it can be emptied and so forth. Furthermore, when adding an order to a rather short route with few visits, there will also be few ways in which the visits can be scheduled. When adding the order to a long route that already consists of many visits, the new visits can be scheduled in many ways. However, for the long routes, the capacity constraints come in play, and especially the maximum capacity of two full containers will prevent many potential routes from being feasible. Still, the addition of one order to an existing route can indeed produce a number of different neighbors and these are all to be considered. This TS algorithm will consider them all and choose the best one for the next iteration.

Removing an order from a route is easier. It can in many cases be done in only one way: By canceling the visits associated with the order or by modifying them so that they do not have anything to do with that order anymore. This includes canceling pickups and deliveries, and modifying load data on the route. The only exception to this would be the removal of an order that is involved in an annihilation procedure. By removing such an order, the other order taking part in the annihilation would either lose the source or the final destination of its empty container. In that case, a small repair is needed.

For example, if a delivery order loses the source of its empty container, a new pickup location is needed. There could be many options as to where to pick up the container and they would all be considered in order to find the best one. Since this is done from a cost minimization point of view, the question is whether it can be picked up at another customer who is not currently involved in any annihilation, if it can be picked up at an existing depot visit, or if it is necessary to set up a new depot visit. Introducing a new annihilation would be cheapest as it would save container handling time, but this is not always possible. The cheapest alternative is to use an existing depot visit. This would cost some additional container handling time but not increase the driving time. However, it may result in a violation of the load constraints at some point later on the route. In that case, the last option would be to introduce a new depot visit. Decisions regarding which depot to visit and when to do it would then also have to be made.

Similarly, a pickup order losing the final destination of its container would need a new destination. This could either be another customer, thereby introducing a new annihilation procedure, an existing depot visit, or a new depot visit. The considerations regarding costs and benefits are the same as when determining a new pickup location.

In conclusion, when a neighborhood is constructed, *all* feasible neighbors are generated and evaluated and the best one is picked for the next iteration.

The Tabu List

With this definition of the neighborhood function, every neighboring route is generated by either adding or removing one order to or from a previously generated route. After an order has been removed from the route, it will be considered a tabu to reinsert that order into the route again; even if the actual visits would be planned differently than before. Similarly, after adding an order to a route, removing it again is a tabu.

The length of the tabu list is set to $\sqrt{n}$ where $n = |O|$ denotes the number of orders in the problem. This definition has been used previously in the literature for VRPs (for eg. **?**]) with good results and it also worked well in preliminary tests. It is a common aspiration criterion that a tabu solution must be better than the best solution seen so far in order for it to be chosen. This criterion is adopted as the only aspiration criterion for this TS heuristic.

Start Solution

The TS needs a start solution. Every time the pricing problem is to be solved, the RMP has just been solved. The optimal dual solution is an input to the TS, but the primal solution to the RMP can also be used by the TS. All routes that have been assigned a non-negative value in the optimal LP solution are also given as input to the TS. These routes are all assumed to have some good features since they are selected in the optimal LP solution to the RMP. They will therefore be used as start solutions for the TS. In addition, since the purpose of the TS is to quickly find a new route with negative reduced cost, the TS algorithm will return the first new route it finds that has a negative reduced cost.

The TS heuristic has two stopping criteria. The first stopping criterion is that a column with a sufficiently large negative reduced cost has been found as mentioned above. The other stopping criterion is that no improvement has been made to the current solution to the pricing problem within a certain number of iterations. As previously mentioned, the aim of the TS solver is to quickly find a good new route for the RMP. If a new route with a negative reduced cost has not been found and

the algorithm has performed a certain number of iterations without finding better solutions, the search will be restarted using a new start route.

Now and then, the TS will start from a route with no orders; an empty route. Using such a start route lets the algorithm decide on its own which region of the solution space that it wants to or needs to explore. The chosen region might not be reachable from any of the available regular start routes.

3.2. Elements from the Adaptive Large Neighborhood Search

A truck can carry up to two full containers at a time. A visit to a dump site will therefore consist of emptying either one or two containers. It is clear that using a dump site visit to empty two containers is beneficial compared to visiting the dump site twice on the route and emptying only one container during each of these visits. When the neighborhood function generates and evaluates all neighbors that can be generated by removing a single order, it can sometimes make a route a lot shorter if it removes a few visits. It will always remove the customer visit, but it will rarely remove the depot visit since this is likely to be related to many orders. The visit at the dump site is removed if there is only one order associated with it. If there are two, the dump site visit will simply be modified but the truck will still have to go there. It is often the case that removing either of the two orders that are using the same dump site visit can save a little time, but it will not always make any significant difference in the reduced cost. However, in the case where two containers are emptied during the same visit at a remote dump site, removing both of them could potentially result in a good route. The situation is illustrated in Figure 5.

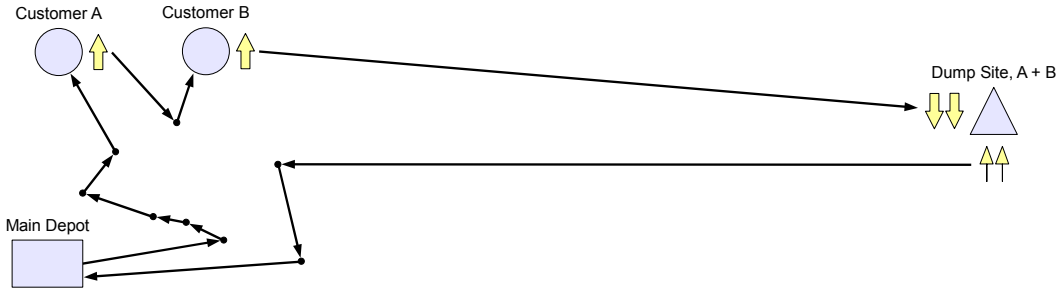

Figure 5: A route with 11 visits of which the visit to customer A and B and their related dump sites are highlighted by larger nodes. It shows a situation where removing only one of the orders placed by customers A and B makes a small difference. However, removing both at the same time would result in a route with no long trips.

Unfortunately, this is not possible with the defined neighborhood function.

We therefore extend the neighborhood with an operator allowing us to make a change that relates specifically to the dump sites. This new operator is inspired by the concept of a destruction method for the Adaptive Large Neighborhood Search (ALNS) as it is presented by **?**]. The ALNS is a metaheuristic framework based upon the idea of having a number of different operators to (partially) destroy a current solution and then another set of operators to rebuild the partial solution thereby constructing a new feasible solution. Part of the strengh of the concept is that the individual operators can focus on different characteristics of the solution.

In **?**] a range of operators for destroying as well as rebuilding solutions for different routing problems is described. The *site removal operator* that we introduce here with specific focus towards the removal of dump sites is based on ideas from that paper.

The site removal operator selects a random dump site visit and removes both orders that have their containers emptied at that visit. Furthermore, these orders are added to the tabu list in a random order. The operator will always pick a site visit where two containers are emptied if such a visit exists on the route. Otherwise, it will pick a random dump site visit where only one container is emptied. If the route consists only of delivery orders, the site removal operator will not change anything. This operator will not be used as often as the standard neighborhood function. Only when the regular TS has not found any improvement for a certain number of iterations, the site removal operator will be applied.

3.3. Initiation and Termination of the master problem

The overall algorithm only has two simple stopping criteria. The first is an upper time limit. When this limit is reached, the algorithm stops. The second stopping criterion is introduced to ensure that the algorithm stops if it is stuck in a local minimum. This criterion is defined as an upper limit on how much time the algorithm may spend without being able to improve the LP solution at all.

To allow for the column generation process to start we need to initialize the master problem with columns that results in a feasible solution. This way the LP can be solved and the first set of dual values produced. An easy and straightforward approach is to enumerate two types of routes: routes with one and two orders. Initial experiments reveal that it is beneficial to add randomly generated routes to the start. The number of routes generated has been made dependent on the total number of orders on the route to control the number of randomly generated routes in the solution process.

3.4. Finding An Integer Solution

What remains is a method to find an integer solution before the algorithm terminates. A large number of routes will be available after having found the optimal solution to the RMP. Since we have already sacrificed optimality by only solving the pricing problem heuristically we have resorted in changing all variables of the RMP to become binary and then solve the corresponding MIP problem using a commercial solver.

The conversion from an optimal LP solution to an optimal integer solution took less than one minute and in most cases less than 30 seconds.

The hybrid CG heuristic has now been presented. Figure 6 shows a flow chart representing the whole algorithm. For more details on the solution approach the reader is referred to **?**].

4. Experimental setup

In this section, all test results are presented and discussed with a focus on the performance of the heuristic. We have been able to compare our solution approach to the method developed for the problem by Transvision (this algorithm will be denoted the commercial solver throughout these tests). The commercial solver is based on a purely metaheuristic approach. It should also be noted that the commercial solver is a very general routing framework. Via parameters and functions different functionality can be turned on or off and as such it is a flexible platform for solving not only RORORproblems but also many other RVRP problems. Due to this flexibility, extensive testing is necessary to get the optimal performance and the cost of flexibility is also that performance will not be as high as for a tailored approach to the problem.

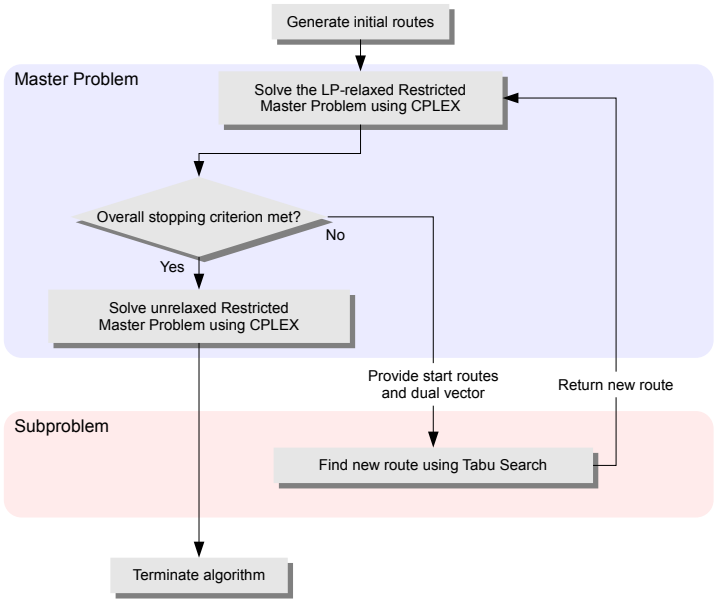

Figure 6: A flow chart representing the hybrid CG heuristic.

Furthermore, some of the characteristics for the real life problem have been removed in this study. In the real-world problem the fleet of trucks is heterogenous, while we assume it is homogeneous. There are other characteristics that have also been relaxed.

This has to be kept in mind when examining the comparison of the two approaches. Also it should be noted that for our comparison Transvision has used the parameter setting they are using for the real-life problem.

4.1. Data Sets

Four different instances have been provided by Transvision. These instances represent four real-world problem instances. Each problem instance contains a set of orders, a set of terminals, the types of which will be identified later, the number of trucks available, as well as start and end times of the workday. For each problem instance it is also specified which depot is considered the main depot, where all routes start and end. In addition, we have generated five additional instances to make the test more broad on size and characteristics of the instances. These instances are constructed based on the data from the real-world instances. Table 1 lists the details of the nine instances we have available; first the four real-world and then the five simulated instances. The two problem instances named B58a and B58b are based in the same geographical region. It should be noted that the generated dataset are produced by randomly selecting orders from the instances B58a and B58b.

Each of the four real-world instances is solved twice by Transvision using their commercial solver. The solution time is fixed to one hour for each problem. These solutions are used as benchmarks when the four problems are solved using the hybrid CG heuristic developed. The time limit will also be set to one hour for this algorithm. In order to be able to assess the consistency of the hybrid CG heuristic, the problems are solved eight times each. We have not had direct access to Transvision's algorithm and therefore the constructed instances are only solved by the hybrid CG heuristic. The instances are solved on an Intel Core2 Duo 1.83 GHz (using only 1 core for the subproblem but possibly 2 when CPLEX solves the RMP) with 2 GB ram, running Windows XP 32 bit.

Name	Orders	Trucks	Workday
A68	68	13	06.30 - 17.30
B58a	58	13	05.00 - 17.30
B58b	58	12	06.00 - 17.30
C48	48	14	06.30 - 16.00
B68a	68	13	05.00 - 17.30
B68b	68	13	05.00 - 17.30
B80a	80	13	05.00 - 17.30
B80b	80	16	05.00 - 17.30
B100	100	18	05.00 - 17.30

Table 1: Main characteristics of the nine problem instances.

This solver is also given a time limit of one hour, but each problem is now solved eight times. The algorithm will stop after 30 minutes with no improvement of the LP solution.

4.2. Experimental results

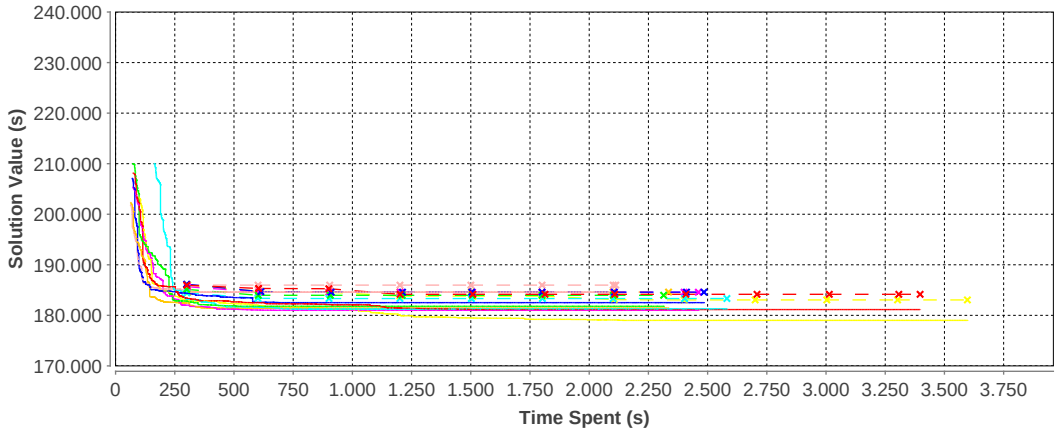

Figure 7: B58b solved using the hybrid CG heuristic. The problem is solved eight times represented by the eight colors. The solid lines denote the optimal LP solutions while the x's are integer solutions.

In order to be able to better assess the performance of the hybrid CG heuristic, the five constructed instances are solved. The main purpose of these tests is to see if the algorithm behaves differently for other instances of size 68 (orders) or for even larger instances. Furthermore, the purpose of solving B80a and B80b is to see if additional trucks can have any effect on the solution process, as beside the number of trucks, the two instances are identical. The solutions found are reported in terms of key values in Table 3. As for the real-world instances, plots illustrating the solution processes are also created for these simulated instances.

Table 2 presented the values of the best solutions to the four real-world problem instances found by the two algorithms. For all four instances, the hybrid CG heuristic was able to find better solutions than the commercial solver. Table 4 lists the average solution values for the two algorithms as well as the absolute and relative improvements. The values of the solutions found by the hybrid CG heuristic are between 8% and 21% lower than those found by the commercial solver and are thus clearly better.

Name	Seed No.	Total Time	Driving Time	Total Distance
A68	1	54h 03m	28h 50m	1661 km
	2	52h 55m	27h 41m	1595 km
B58a	1	67h 09m	40h 09m	2313 km
	2	63h 03m	35h 23m	2038 km
B58b	1	63h 06m	34h 06m	1965 km
	2	62h 32m	34h 32m	1989 km
C48	1	38h 04m	16h 24m	927 km
	2	37h 21m	16h 00m	910 km
Name		**Total Time**	**Driving Time**	**Total Distance**
A68	Best	44h 16m	20h 01m	1153 km
	Worst	46h 44m	22h 37m	1302 km
	Average	45h 25m	21h 20m	1229 km
B58a	Best	50h 59m	26h 29m	1525 km
	Worst	51h 47m	26h 47m	1543 km
	Average	51h 22m	26h 42m	1538 km
B58b	Best	50h 51m	25h 31m	1470 km
	Worst	51h 40m	26h 00m	1497 km
	Average	51h 11m	25h 35m	1474 km
C48	Best	34h 14m	14h 14m	819 km
	Worst	35h 04m	14h 54m	858 km
	Average	34h 41m	14h 22m	828 km

Table 2: Top-half is the values of the best solutions to the real-world instances using the commercial solver, and the bottom half is the hybrid CG heuristic. The commercial solver was run with two seeds specified by Transvision.

The commercial solver has only been applied twice for each problem instance. Assessing the consistency of the solutions that it can find is therefore hard. Even though it seems very unlikely that it would be able to beat the hybrid CG heuristic for B58a and B58b, it might be possible for C48. However, in that case, the solution values of the commercial solver would vary much more than those of the hybrid CG heuristic. The solution values between 37 and 38 hours found by the commercial solver for C48 correspond to more than 134,000 seconds. If the commercial solver should be able to find solutions that can compete with the average value of 34 hours and 41 minutes of the solutions found by the hybrid CG heuristic, it would not be able to compete on consistency.

Figure 7 illustrates the solution processes for the B58b real-world instance for our hybrid CG algorithm. The corresponding graphs for the other real-world instances look similar. In general, the commercial solver improves the solution values at a relatively constant speed. The hybrid CG heuristic improves the solution value much faster during the first five minutes, and then it slows down and barely improves the solution for the rest of the time. During the last 30 minutes, the improvements to the solution value are very small. The average solution values for both algorithms after five minutes for all four real-world problem instances are given in Table 5 and are compared to those obtained after an hour.

Name		Total Time	Driving Time	Total Distance
B68a	Best	63h 46m	33h 31m	1931 km
	Worst	64h 36m	34h 21m	1979 km
	Average	64h 16m	34h 11m	1969 km
B68b	Best	55h 12m	26h 47m	1542 km
	Worst	56h 41m	27h 26m	1580 km
	Average	56h 00m	27h 05m	1560 km
B80a	Best	67h 04m	34h 34m	1991 km
	Worst	68h 06m	35h 26m	2041 km
	Average	67h 38m	34h 55m	2011 km
B80b	Best	67h 42m	34h 42m	1998 km
	Worst	68h 06m	35h 06m	2022 km
	Average	67h 50m	34h 53m	2010 km
B100	Best	83h 41m	40h 41m	2343 km
	Worst	84h 58m	41h 48m	2408 km
	Average	84h 30m	41h 30m	2390 km

Table 3: The values of the best solutions to the five constructed instances.

	Total Time		Improvement	
Name	Commercial	Hybrid CG	Absolute	Relative
A68	53h 29m	45h 25m	8h 04m	15.1 %
B58a	65h 06m	51h 22m	13h 44m	21.1 %
B58b	62h 49m	51h 11m	11h 38m	18.5 %
C48	37h 43m	34h 41m	3h 02m	8.0 %

Table 4: The average solution values from the hybrid CG heuristic compared to the average solution values from the commercial solver. Both algorithms have had a time limit of one hour.

The commercial solver improves the solution values a lot more during the last 55 minutes than the hybrid CG heuristic does. However, it is worth noting that the integer solutions found by the hybrid CG heuristic after five minutes are already better than those found by the commercial solver after one hour for all four instances. Table 4 compares the values of the solutions found by the two algorithms after an hour, and Table 6 makes a similar comparison for the solutions found by the two algorithms after only five minutes. The values of the solutions found by the hybrid CG heuristic are between 11% and 27% lower than those found by the commercial solver when the time limit is only five minutes. There is thus no doubt that the CG heuristic outperforms the commercial solver.

The hybrid CG heuristic often stops before the 60 minutes have passed because no improvement has been made for more than 30 minutes. Sometimes, the number of start routes in a solution is constant for a long time and remains at this number for the rest of the time. These are also the test runs that have been aborted early. The algorithm keeps switching back and forth between the subproblem and the RMP and gets to try all the available start solutions without being able to generate a new route with negative reduced cost.

For the constructed instances we observe that the solution value improves significantly during the first five minutes and then the improvement rate decreases to almost

zero. However, there is a clear tendency that it takes longer for the improvement rate to decrease for the large instances (see Figure 8).

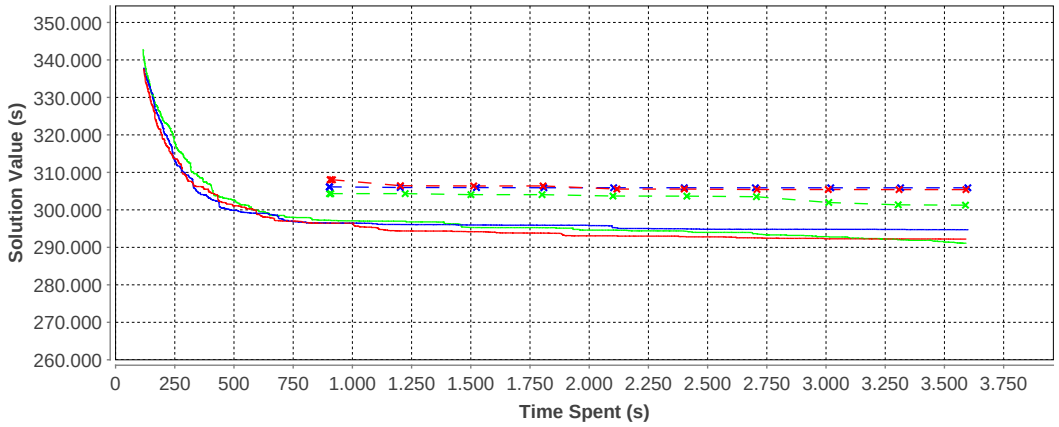

Figure 8: Instance B100 solved using the hybrid CG heuristic. The problem is solved three times represented by the three colors. The solid lines denote the optimal LP solutions while the x's are integer solutions. There do not always exist feasible integer solutions during the first 15 minutes of solving. Therefore, integer solutions are not found until the first 15 minutes have passed.

It is not surprising that solving larger instances requires more time. But it is interesting to see that the TS of the subproblem solver of the algorithm is not really slowed down by the increased size of these instances. Figures 9 and 10 show the number of iterations performed as a function of the time the algorithm has run for B58b and B100. The time it takes to perform an iteration does not seem to depend much on the size of the problem. However, for a large problem the algorithm keeps going for a longer time performing more iterations with significant improvements to the solution value. This makes sense, as there is a larger and more complex search space to investigate. This is the reason why the improvement rate slows down a little later for the larger instances.

5. Conclusion

The purpose of this paper was to assess the potential of CG based algorithms for solving the RORORP problem.

The RORORP problem has been presented and defined as a generalized set partitioning problem. From this formulation it was apparent that the problem could potentially be solved by an algorithm that is based on a CG scheme, but it was also clear from the complexity of the pricing problem that the possibility of solving the

	Commercial		**Hybrid CG Heuristic**	
Name	5 Minutes	60 Minutes	5 Minute Runs	60 Minute Runs
A68	57h 30m	53h 29m	47h 25m	45h 25m
B58a	72h 39m	65h 06m	52h 56m	51h 22m
B58b	70h 50m	62h 49m	51h 39m	51h 11m
C48	$\approx$ 39h 30m	37h 43m	34h 50m	34h 41m

Table 5: The average value of the best known objective function values after 5 and 60 minutes for both algorithms. The values given refer to route duration.

	Total Time		Improvement	
Name	Commercial	Hybrid CG Heuristic	Absolute	Relative
A68	57h 30m	47h 25m	10h 05m	17.5 %
B58a	72h 39m	52h 56m	19h 43m	27.1 %
B58b	70h 50m	51h 39m	19h 11m	27.1 %
C48	≈ 39h 30m	34h 50m	≈ 4h 40m	≈ 11.8 %

Table 6: A comparison of the solutions found after five minutes.

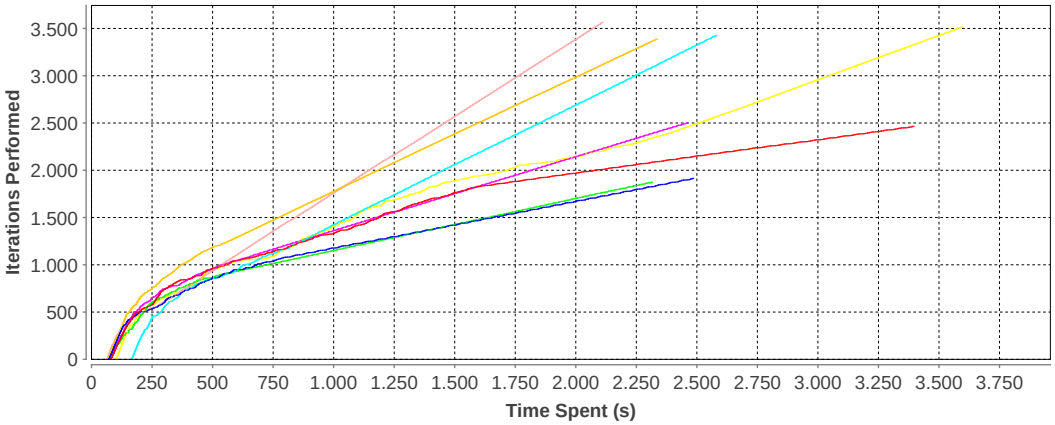

Figure 9: The number of iterations performed as a function of the time spent for B58b.

problem to optimality within reasonable time was non-existent. A solution framework based on CG and TS was therefore presented.

The algorithm was tested on nine different problem instances, of which four were real-world instances provided by Transvision. These four problem instances contained between 48 and 68 orders. The other five instances were simulated from the four real-world instances. These instances contained between 68 and 100 orders.

It can be concluded that the implementation of the hybrid CG heuristic was able to find feasible solutions to all nine problem instances. The algorithm was able to find good feasible solutions within 15 minutes for instances with up to 100 orders.

Only the four real-world instances were solved using the commercial solver from

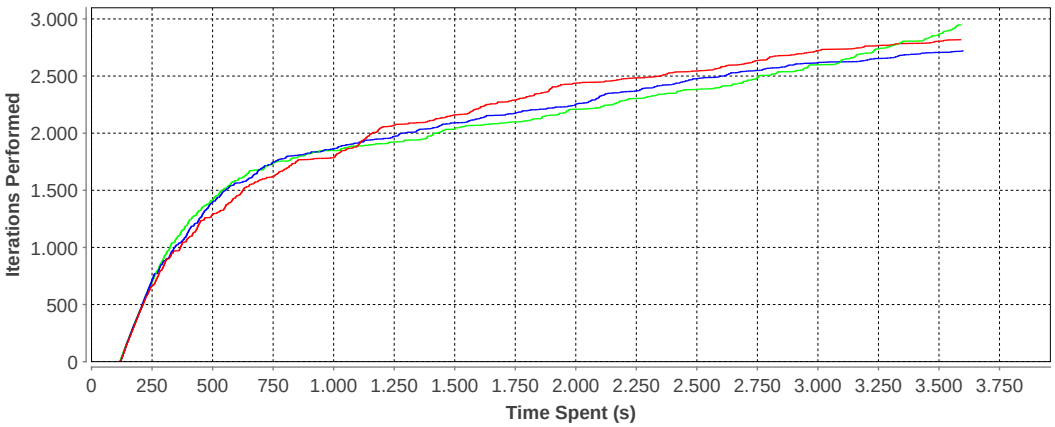

Figure 10: The number of iterations performed as a function of the time spent for sB100.

Transvision for comparison. With a time limit of an hour, the hybrid CG heuristic outperformed the purely heuristic algorithm for all four real-world instances. The values of the solutions found by the CG heuristic were between 8% and 21% lower than those found by the purely heuristic algorithm. It was also evident from the results that the hybrid CG heuristic was much faster than the purely heuristic algorithm. When the algorithms only had five minutes to solve the problem, the hybrid CG heuristic found feasible solutions with values that were between 11% and 27% lower than the values of the solutions found by the purely heuristic commercial solver. It should further be noted that the solutions found by the CG heuristic after five minutes were better than those found by the commercial solver after an hour.

Finally, it should be noted that the hybrid CG algorithm allows for the commercial solver of Transvision to be used as the algorithm for solving the pricing problem. This would allow to exploit the larger flexibility in an industrial software to cater for different special cases, company-specific rules or different legislation.

[] Archetti, C., Bouchard, M., & Desaulniers, G. (2011). Enhanced branch and price and cut for vehicle routing with split deliveries and time windows. *Transportation Science*, *45*, 285–298. doi:10.1287/trsc.1100.0363.

[] Archetti, C., & Speranza, M. G. (2004). Vehicle routing in the 1-skip collection problem. *Journal of the Operational Research Society*, *55*, 717 – 727.

[] Baldacci, R., Bodin, L., & Mingozzi, A. (2006). The multiple disposal facilities and multiple inventory locations rollon-rolloff vehicle routing problem. *Computers & Operations Research*, *33*, 2667–2702.

[] Blanc, I. l., Krieken, M. v., Krikke, H., & Fleuren, H. (2006). Vehicle routing concepts in the closed-loop container network of ARN - A case study. *OR Spectrum*, *28*, 53–71.

[] Bodin, L., Mingozzi, A., Baldacci, R., & Ball, M. (2000). The rollon-rolloff vehicle routing problem. *Transportation Science*, *34*, 271 – 288.

[] Chabrier, A. (2006). Vehicle routing problem with elementary shortest path based column generation. *CAOR*, *33*, 2972 – 2990.

[] De Meulemeester, L., Laporte, G., Louveaux, F. V., & Semet, F. (1997). Optimal sequencing of skip collections and deliveries. *Journal of the Operational Research Society*, *48*, 57 – 64.

[] Desaulniers, G., Lessard, F., & Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, *42*, 387–404.

[] Drexl, M. (2012). Rich vehicle routing in theory and practice. *Logistics Research*, *5*, 47–63.

[] Golden, B. L., Assad, A. A., & Wasil, E. A. (2002). The vehicle routing problem. chapter Routing vehicles in the real world: applications in the solid waste, beverage, food, dairy, and newspaper industries. (pp. 245 – 286). SIAM.

[] Guan, C.-h., Cao, Y., & Shi, J. (2010). Tabu search algorithm for solving the vehicle routing problem. In *Proceedings of the 2010 Third International Symposium on Information Processing* (pp. 74–77).

[] Hauge, K. M. (2011). *A Roll-On-Roll-Off Vehicle Routing Problem with Increased Capacity*. Master's thesis Technical University of Denmark. Classified (but partially available by contacting the corresponding author of this article).

[] Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers and Operations Research*, *34*, 2403–2435.

[] Prescott-Gagnon, E., Desaulniers, G., & Rousseau, L.-M. (2009). A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, *54*, 190–204.

[] Qureshi, A. G., Taniguchi, E., & Ymada, T. (2009). Column generation-based heuristics for vehicle routing problem with soft time windows. *Journal of the Eastern Asia Society for Transportation Studies*, *8*, 1 – 15.

[] Schmid, V., & Doerner, K. F. (2010). Survey: Matheuristics for rich vehicle routing problems. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6373*, 206–221.

[] Wy, J., & Kim, B.-I. (2013). A hybrid metaheuristic approach for the rollon-rolloff vehicle routing problem. *Computers and Operations Research*, *40*, 1947 – 1952.

[] Wy, J., Kim, B.-I., & Kim, S. (2013). The rollon-rolloff waste collection vehicle routing problem with time windows. *European Journal of Operational Research*, *224*, 466 – 476.